Information & Security

B. Gdowski, R. Kościej & M. Niemiec
vol. 50, no. 1 (2021): 23-36
https://doi.org/10.11610/isij.5010

Published since 1998
ISSN 0861-5160 (print), ISSN 1314-2119 (online)

Research Article

Heuristic-based Intrusion Detection Functionality in a Snort Environment

***Bartłomiej Gdowski, Rafał Kościej, Marcin Niemiec* (✉)**

AGH University of Science and Technology
Mickiewicza 30, 30-059 Krakow, niemiec@agh.edu.pl

ABSTRACT:

This article provides an introduction to intrusion detection systems, focusing on extending the Snort environment's functionalities by adding a new heuristic detection algorithm. The algorithm allows to detect selected types of cyberattacks through analysis of received packets and based on a list of malicious Internet Protocol addresses. Furthermore, the algorithm underwent functional verification. The results confirmed that the algorithm successfully detects the packets originating from the provided list and rates them accordingly.

ARTICLE INFO:

RECEIVED: 10 JUNE 2021
REVISED: 02 SEP 2021
ONLINE: 18 SEP 2021

KEYWORDS:

cybersecurity, intrusion detection, network attack, heuristic algorithm

1. Introduction

The constant evolution of science and technology leads to the proliferation of challenges in our everyday lives. With the Internet becoming one of the most important inventions in the recent century and obviously an integral part of today's world, new threats have emerged. More specifically, the evolution of the cyber world is associated with the parallel evolution of threats in cyberspace.

The global network has become a part of our life. People more and more often use it to do the things that traditionally would be done in person. Convenient online payments (for bills, clothes, or even food) have convinced many to

✉ Corresponding Author: Tel.: +48 126174803 ; E-mail: niemiec@agh.edu.pl

go online, even for the simplest activities. That meant a need to develop more and more functions available for users. As mentioned earlier, every coin has two sides – linking our everyday lives to the Internet meant that someone will try to take advantage of careless users. Harmful software, viruses, and many more ways to hack anything on the web are developed daily. This means that network security has become a major issue – being safe in the real world is not enough; we also need to be careful on the Internet.

Each user wants to feel safe while performing any action online and, of course, to be safe whilst doing it. That means there is a big responsibility on the Internet providers, as they supply the Internet's users with a gateway to the outside world. This responsibility becomes even more significant when it comes to the security of bigger companies – breaches in security could mean loss of personal data (email addresses, logins, passwords, even credit cards details, and many more). Vulnerabilities could mean losses for a single user but also could mean thousands, millions, or even billions of dollars. That is the reason why the evolution of threats causes the development of respective security tools.

The variety of available tools allows them to be used by big companies, as well as single users. In this article, the authors concentrate on the development of a heuristic-based functionality that would be compatible with one of the well-known Internet Detection and Prevention Systems (IDPS) available on the market. The results obtained in this paper are a part of the prototype developed in the *H2020 ECHO* project. The European network of Cybersecurity centres and competence Hub for innovation and Operations (ECHO) [1] is a project aimed at improving the European Union's cybersecurity throughout developing new solutions to cyber problems. One of the project's prototypes is the *SNORT Module*.

2. Intrusion Detection Systems

An Intrusion Detection System (IDS) is a software application or a device capable of monitoring systems (hosts) or network traffic and detecting anomalies. Malicious activity is either collected on the device or sent externally (e.g., to an administrator of the network). There are many types of IDSs. They can be classified into two main categories, which are presented below.[2,3,4]

- Classification by the analysed activity – in other words, placement of an IDS.
  - Host IDS (HIDS) – a system that protects a single device system files by monitoring packets sent to and by the device whilst operating on a database of system objects. Every time an anomaly is detected, HIDS takes a snapshot of the current state of monitored files. Then the files are compared to the database – if files were modified or deleted, an alert is sent to the administrator.
  - Network IDS (NIDS) – a system that protects a network by monitoring inbound and outbound packets from every device in the specified network. NIDS analyses the web traffic by matching packets to a library of known attacks or using heuristic techniques (e.g., an algorithm). It is capable of logging and/or alerting when the threat is found.

- Classification by the detection method, i.e., the way the anomalies are detected.
  - Signature-based IDS (also called definition-based or misuse-based) – this type of system operates on a database of known vulnerabilities or attack patterns. It works similarly to anti-virus software. Signature-based IDSs work very well with detecting known attacks, but they are not capable of detecting new attacks. This means that it is crucial that the producer/vendor of the product updates the threats database frequently.
  - Anomaly-based IDS (also called behaviour-based) – this type of system operates on a similar pattern to that of HIDS. First, it identifies the normal behaviour of the network by a performance baseline under normal operating conditions. Next, it constantly compares current network behaviour against the network's baseline. Every time an anomaly is detected, an alert indicating a potential attack is sent. An anomaly-based IDS is very effective at finding zero-day exploits, but at the same time, may be flooding with false positives.

The placement of an IDS is critical and varies depending on what the user wants to protect – e.g., to protect one critical device in the network, one can use either HIDS, NIDS, or even both of them, depending on available resources. Therefore, it is crucial to keep the balance between the network performance and the range of IDS operations.

The most obvious placement of an IDS is behind the firewall – in this way, the whole network can be monitored, but at the same time, this might create a bottleneck that would decrease the overall throughput of the network. On the other hand, if the IDS is placed deeper inside the network, the performance will be maintained, while a part of the network will be left vulnerable.[5]

3. Snort Environment

Snort is a free open source NIDS capable of logging and/or analysing incoming traffic in real-time on IP networks. It was originally released in 1998 as a cross-platform network sniffing tool by Martin Roesch. With the help of the community members, it evolved into a powerful Intrusion Detection and Prevention System (IDPS). It is a great example of how successful an open-source tool can be when developed in cooperation with the users, for example, by reporting and even fixing bugs or contributing to the source code.[6]

The development of *Snort* has been coordinated by *Sourcefire*, a company founded in 2001 by Roesch – the software's creator. *NSS Labs*, a company specialized in testing software and hardware to reveal vulnerabilities to cyber-threats, has awarded an *Approval Certificate* to *Snort* versions: 1.8.1, 1.8.6, and 2.0.[7] In 2009 it was called one of "the greatest open source software of all time" by *Info World*.[8] *Snort* is now developed by *Cisco*, which purchased *Sourcefire* in 2013. The tool itself is free, but it is based on a paid subscription model – the newest threat rules are available for subscribers immediately, while free users get access to rules after 30 days.[6,9]

Snort operates in one of three modes described below.[9]

- Sniffer Mode – reads packets and displays them in a stream on the screen. The user themself can check the packed content. The most basic version displays only Transmission Control Protocol (TCP)/Internet Protocol (IP) packet headers, but it can be configured to also show User Datagram Protocol (UDP)/Internet Control Message Protocol (ICMP) headers and packet data.
- Packet Logger Mode – used for collecting logs. It starts automatically, overriding the sniffer mode when the user specifies logging directory. There is a possibility to configure packet logger mode to save split logs of different hosts to subdirectories of the log folder. It also allows to save the logs as a binary file—useful when saving logs from a high-speed network—and to open saved logs (with filters) using the *Snort* console.
- Network Intrusion Detection Mode is the most complex mode; it performs multistage detection and analysis on network traffic. The main feature of *Snort* as NIDS is using flexible rule language to describe specific traffic to be collected by the software. The process of collecting and processing packets through the *Snort* engine will be described later in this article.

Snort's engine consists of a sniffer (packet acquisitor and decoder), preprocessors, detection engine, and the output responsible for generating alerts.[9,10]

The first step of detecting an unwanted anomaly is obviously collecting (sniffing) network traffic and identifying the structure of each packet. It means that there is a packet capture and a filtering engine needed to acquire data such as:[11]

- the packet capture time;
- length of the packet;
- size of the captured packet;
- a pointer to the contents of the packet.

After capturing the packet, *Snort* begins decoding – the acquired packet enters the packet decoder depending on the link layer from which it is read. Decoding is pretty much the same regardless of the link layer – *Snort* verifies the data and calls into higher layer decoders until there is no higher layer.[12]

Next in the processing queue are the preprocessors. They allow extending the functionality of *Snort* by allowing to configure modules of the packet processing easily. In the end, the processed packet reaches the detection engine, where the rule set—configured by the user—is applied to incoming traffic.

As mentioned earlier, the rule language is very flexible and gives many possibilities to control traffic in various ways. A single rule consists of two parts: header and options. The header contains the rule's action, protocol type (currently supported are: TCP, UDP, ICMP, and IP), destination IP addresses and netmasks, direction operator (used to indicate the direction of the traffic that rule applies to), source and destination ports information. The options section contains alert messages and information that determine if the rule action should be taken depending on the inspected packet.[9]

4. The New Functionality

Snort is a very flexible tool when it comes to adding new functionalities. Plugins introduced in one of its earliest versions – ver. 1.5 [13] – made automation of some actions during packet processing easier. One of the ways to extend the functionality of the program is using either a detection plugin or a preprocessor. This section introduces the latter, as the detection algorithm will operate in cooperation with *Snort* with an additional preprocessor.

The preprocessors are a part of *Snort* that is crucial when it comes to developing a new functionality inside the environmental engine. There are two options to do so: rewriting the existing preprocessor or writing a new preprocessor from an existing template. The latter being, of course, more complicated, or – simply – difficult. Why is there a need for new functionalities when there are so many existing ones? The answer is very straightforward – for the same reasons why popular preprocessors were written in the first place.

These major reasons are:

- reassembling packets – sometimes defragmentation (frag3) and reassembly (stream) of the packets are not sufficient to detect specific attacks, so some functions to split or combine packets in specific ways could be implemented;
- decoding and normalizing protocols – *Snort* supports decoding and normalization for different protocols (e.g., Telnet, Hypertext Transfer Protocol (HTTP) or File Transfer Protocol (FTP)), but it is hard to avoid attacks on different versions of these protocols (on the other hand, it is possible to configure existing decoding protocols in such way that they will be less dependent on the protocol version, or even allow pattern-matcher to skip the negotiation data, e.g., in the case of Telnet);
- non-rule or anomaly-based detection – apart from a very efficient rule system, *Snort* also depends on automation, so existing preprocessors can be used to catch specific attacks instead of writing lots of rules (e.g., *sfPortscan* looking for scan attacks);
- and finally, probably the most important reason – developing new functionalities to address the cybersecurity challenges.

The authors propose a new preprocessor to use with *Snort* called *Heuristic*. It is still in development; however, the research presented here is being conducted on the first stable version (alpha). The functions of the preprocessor will be covered in the "Verification" section. In this section, the authors would like to focus on the process of adding a new preprocessor to the *Snort* environment.

The alpha version of the *Heuristic* preprocessor is made of two files: spp_*.h and spp_*.c (a header and proper C file) where *spp* refers to its type – in *Snort*, detection plugins have *sp* prefix, while preprocessors (as above) have *spp* prefix. In this case they are called spp_heuristic. The files must be added to snort*/src/preprocessors (where snort* is *Snort's* main catalog). This could be done either by copying the files manually or by using the command:

```
cp PATH_TO_HEURISTIC_FILES /spp_heuristic .*
PATH_TO_SNORT_FOLDER /src/preprocessors
```

Plugins are linked to *Snort* in a static way, so some of *Snort's* files have to be edited before the preprocessor is detected. First, snort*/src/plugbase.c file has to be edited by adding include directive of the plugin's header file (e.g., at the end of *built-in preprocessors* section, as in the Listing 4.1 and Setup() function to preprocessor initialization list (void RegisterPreprocessors() function) – as in Listing 4.2.

```
/* built-in  preprocessors  */
#include "preprocessors/spp_rpc_decode.h"
#include "preprocessors/spp_bo.h"
#include "preprocessors/spp_session.h"
#include "preprocessors/spp_stream6.h"
#include "preprocessors/spp_arpspoof.h"
#include "preprocessors/spp_perfmonitor.h"
#include "preprocessors/spp_httpinspect.h"
#include "preprocessors/spp_sfportscan.h"
#include "preprocessors/spp_frag3.h"
#include "preprocessors/spp_normalize.h"
#include "preprocessors/spp_heuristic.h"
```

Listing 4.1: Including the preprocessor's header into plugbase.c file.

```
void RegisterPreprocessors(void){
    LogMessage("Initializing  Preprocessors !\n");
    SetupARPspoof();
#ifdef  NORMALIZER
    SetupNormalizer();
#endif
    SetupFrag3();
    SetupSessionManager();
    SetupStream6();
    SetupRpcDecode();
    SetupBo();
    SetupHttpInspect();
    SetupPerfMonitor();
    SetupSfPortscan();
    SetupHeuristic();
}
```

Listing 4.2: Including the preprocessor's Setup() function into plugbase.c file.

The next file to edit is preprocids.h in the same folder (src). There are also two things to be done. First is defining an ID number of the preprocessor (shown in Listing 4.3). There is a max of 50 preprocessors inside the base 2.9.16 version of Snort at the same time, so the ID should be between the last preprocessor ID and 50 – preferably the lowest number possible.

```
#define  PP_HTTP2                     35
#define  PP_CIP                       36
#define  PP_MAX                       37
#define  PP_HEURISTIC                 38
#define  PP_ALL                       50
#define  PP_ENABLE_ALL  (~0)
```

Listing 4.3: Defining the ID of the preprocessor in the preprocids.h file.

Second, *Snort* has to know what the preprocessor type is. According to contents of preprocids.h there are three types:

- Network Analysis Policy processing preprocessors – if enabled by the configuration, they are never disabled;
- Firewall and Application ID & Network Discovery preprocessors – same as the previous one;
- Application preprocessors – plugins that are enabled according to the type of processed stream (that is where Heuristic has to be included – edited class presented in Listing 4.4).

```
#define PP_CLASS_PROTO_APP (
(UINT64_C (1)  << PP_BO) | (UINT64_C (1)  << PP_DNS) |
(UINT64_C (1)  << PP_FTPTELNET) | \ (UINT64_C (1)  <<
PP_HTTPINSPECT) | (UINT64_C (1)  << PP_RPCDECODE) | \
(UINT64_C (1)  << PP_SHARED_RULES) | (UINT64_C (1)  <<
PP_SMTP) | (UINT64_C (1)  << PP_SSH) | (UINT64_C (1)  <<
PP_SSL) | (UINT64_C (1)  << PP_TELNET) | (UINT64_C (1)  <<
PP_ARPSPOOF) | (UINT64_C (1)  << PP_DCE2) | (UINT64_C (1)  <<
PP_SDF) | (UINT64_C (1)  << PP_ISAKMP) | (UINT64_C (1)  <<
PP_POP) | (UINT64_C (1)  << PP_IMAP) | (UINT64_C (1)  <<
PP_GTP) | (UINT64_C (1)  << PP_MODBUS) | (UINT64_C (1)  <<
PP_DNP3) | (UINT64_C (1)  << PP_FILE) | (UINT64_C (1)  <<
PP_FILE_INSPECT) | (UINT64_C (1)  << PP_HEURISTIC))
```

Listing 4.4: Adding the preprocessor to appropriate class in preprocids.h file.

According to the ID number chosen in preprocids.h the array of preprocessors has to be updated in snort.c file (PP_HEURISTIC has been added with ID38, so it should be the 38th element of the array – which can be seen in Listing 4.5).

```
static  const  char* preproc [50] = {
   "PP_BO", "PP_APP_ID", "PP_DNS", "PP_FRAG",
   "PP_FTPTELNET", "PP_HTTPINSPECT",   "PP_PERFMONITOR",
   "PP_RPCDECODE", "PP_SHARED_RULES", "PP_SFPORTSCAN",
   "PP_SMTP", "PP_SSH", "PP_SSL", "PP_STREAM",
   "PP_TELNET", "PP_ARPSPOOF", "PP_DCE", "PP_SDF",
   "PP_NORMALIZE", "PP_ISAKMP", "PP_SESSION", "PP_SIP",
   "PP_POP", "PP_IMAP", "PP_NETWORK_DISCOVERY",
   "PP_FW_RULE_ENGINE", "PP_REPUTATION", "PP_GTP",
   "PP_MODBUS", "PP_DNP ", "PP_FILE", "PP_FILE_INSPECT",
   "PP_NAP_RULE_ENGINE", "PP_REFILTER_RULE_ENGINE",
   "PP_HTTPMOD", "PP_HTTP ", "PP_CIP", "PP_MAX",
   "PP_HEURISTIC"};
```

Listing 4.5: Adding Heuristic to the preprocessors array in the snort.c file.

As *Heuristic* (and other preprocessors) is written in C language, it has to be built by the make command. It means that both files (spp_heuristic.h/.c) have to be included in Makefile.am – identically as in Listing 4.6 – inside src/preprocessors folder.

```
libspp_a_SOURCES = spp_arpspoof.c spp_arpspoof.h \
    spp_bo.c spp_bo.h \
    spp_rpc_decode.c spp_rpc_decode.h   \
    spp_perfmonitor.c spp_perfmonitor.h \
    perf.c perf.h \
    perf-base.c perf-base.h \
    perf-flow.c perf-flow.h \
    perf-event.c perf-event.h \
    perf_indicators.c perf_indicators.h \
    spp_httpinspect.c spp_httpinspect.h \
    snort_httpinspect.c snort_httpinspect.h \
    portscan.c portscan.h \
    spp_sfportscan.c spp_sfportscan.h \
    spp_frag3.c spp_frag3.h \
    str_search.c str_search.h \
    spp_stream6.c spp_stream6.h \
    spp_session.c spp_session.h \
    session_api.c session_api.h \
    stream_api.c stream_api.h \
    spp_normalize.c spp_normalize.h \
    sip_common.h cip_common.h \
    spp_heuristic.c spp_heuristic.h
```

Listing 4.6: Adding Heuristic to the preprocessors array in the snort.c file

5. Verification

As mentioned earlier, *Heuristic* adds new functions to *Snort*. The heuristic-based detection can take into account data shared by the federated organisations. The organisations can distribute information about the severity of a threat associated with a given IP address. Therefore, the snort.conf configuration file may contain a path to a *.csv file with unsafe IPv4 addresses and assigned flags. A sample file is included as Listing 5.1.

Each address should have a flag (indicating how dangerous the address is) assigned to it. There are three flags implemented in Heuristic's alpha version:

- M – Malicious – the least dangerous type;
- D – Dangerous;
- C – Critical – the most dangerous type.

Each flag has assigned a value that will be added (the values have to be negative, so their absolute value will be subtracted instead) to the packet rating. Default values are: M:-1; D:-2; C:-3; and can be edited in the configuration file. The above-mentioned evaluation of packets starts at a predefined packet_value variable. Depending on the flag assigned to the address, the packet rating is updated (hence the negative values assigned to the flags). In the end, packet_value is compared to the sensitivity variable, which is a deciding factor in displaying alerts. There are two cases.

```
packet_value + flag_value ≤ sensitivity - an alert is being
displayed;
packet_value + flag_value > sensitivity - no action is performed.
```

It is worth mentioning that the advanced multivariable heuristic detection algorithm, which takes into account different kinds of flags and entropy value was introduced by the authors in the *Entropy* journal.[14] Listing 5.2 represents an example configuration file with all of the *Heuristic* preprocessor variables set.

Listing 5.3 presents a fragment of the *HeuristicSnort* initialization message (with configuration from Listing 5.2). This part of *Snort's* initialization header confirms that the *Heuristic* preprocessor has been initialized, and both snort.conf and HeuristicIPAddr.csv have been read correctly.

```
192.168.2.1,D
192.168.2.57,M
192.168.2.63,C
```

Listing 5.1: Sample *.csv file with malicious addresses

```
#declaration  of  Heuristic  variables
preprocessor  heuristic: sensitivity  14  packet_value  20
preprocessor  heuristic_ip_dangerous:
    #path to the *.csv  file
    filename  PATH_TO_CSV_FILE/HeuristicIPAddr.csv \
    #flag  value  override
    flag C  -15 \
    flag D  -10 \
    flag M -5
```

Listing 5.2: Sample snort.conf file with Heuristic variables set

Listing 5.4 presents the output of *HeuristicSnort*. An alert consists of:

- [Packet number] – the packet number in a given iteration of the program;
- [IP_ADDR]->Source address – source address of the packet;
- [FLAG]X – flag defined in the configuration file;
- [Packet value]:X – value of that particular packet.

```
Heuristic global config:
    Sensitivity: 14
    Start packet value: 20
Heuristic IP dangerous config:
    IP ranking filename: PATH_TO_CSV_FILE/HeuristicIPAddr.csv
    IP ranking record number 3
    Flags value: D: -10, M: -5 C: -15
```

Listing 5.3: Confirmation of Heuristic initialization.

To fully understand the *HeuristicSnort 's* output from Listing 5.4, it has to be known what devices are represented by the addresses from Listing 5.1.

- 192.168.2.1 is a router address (hence the number and frequency of incoming packets).
- 192.168.2.57 is a local IP address (the M flag updates the packet_value to 15, which is bigger than sensitivity, so none of the packets from this address were alerted).
- 192.168.2.63 is an address of a virtual machine (VM), which executed a ping command twice (each time 2 packets).

```
[177][ IP_ADDR ] - >192.168.2.1 , [FLAG]D, [Packet  value ]:10
[178][ IP_ADDR ] - >192.168.2.63 , [FLAG]C, [Packet  value ]:5
[179][ IP_ADDR ] - >192.168.2.1 , [FLAG]D, [Packet  value ]:10
[180][ IP_ADDR ] - >192.168.2.63 , [FLAG]C, [Packet  value ]:5
[181][ IP_ADDR ] - >192.168.2.1 , [FLAG]D, [Packet  value ]:10
[192][ IP_ADDR ] - >192.168.2.1 , [FLAG]D, [Packet  value ]:10
[193][ IP_ADDR ] - >192.168.2.1 , [FLAG]D, [Packet  value ]:10
[194][ IP_ADDR ] - >192.168.2.63 , [FLAG]D, [Packet  value ]:5
[195][ IP_ADDR ] - >192.168.2.1 , [FLAG]D, [Packet  value ]:10
[196][ IP_ADDR ] - >192.168.2.63 , [FLAG]C, [Packet  value ]:5
[197][ IP_ADDR ] - >192.168.2.1 , [FLAG]D, [Packet  value ]:10
[198][ IP_ADDR ] - >192.168.2.1 , [FLAG]D, [Packet  value ]:10
[199][ IP_ADDR ] - >192.168.2.1 , [FLAG]D, [Packet  value ]:10
[286][ IP_ADDR ] - >192.168.2.1 , [FLAG]D, [Packet  value ]:10
[288][ IP_ADDR ] - >192.168.2.1 , [FLAG]D, [Packet  value ]:10
[313][ IP_ADDR ] - >192.168.2.1 , [FLAG]D, [Packet  value ]:10
```

Listing 5.4: Fragment of Heuristic_Snort output.

The contents of Listing 5.4 and information below that listing confirm that the *HeuristicSnort* is working properly and generates alerts for every address that violates the preprocessor's "policy."

6. Summary

The authors of this paper have provided an overview of types of IDSs and an introduction to *Snort* – one of the tools used in intrusion detection. Then, they provided information on each step of adding a new preprocessor to *Snort's* environment. Finally, the functionalities of the *Heuristic* preprocessor have been presented and verified in a short test. It has been proved that the algorithm successfully rates each packet depending on its source address and assigned flag.

New functionality is in its early development stage – as the alpha version was presented in this paper. The configuration of the preprocessor makes it possible to easily modify existing flags. It is also possible to add new flags analogically. The simple layout of input *.csv file allows the user to add, remove or modify existing records on-the-go. The packet value calculation influences the decision whether or not a specific packet should be reported. Summing everything up, the algorithm is flexible and can be adapted to specific network environments, e.g., hospitals or server rooms. The flexibility of the algorithm means that it can be used anywhere, depending on the needs and requirements.

The design of an innovative detection algorithm and development of a new functionality in the *Snort* environment was related to the work in the European research project – *H2020 ECHO*. This project was initiated by the European Commission in 2019 and consists of over 30 partners from different sectors. As a part of the project, tools and prototypes are developed to help increase network security.

Cybersecurity is an area of constant development. Therefore, the work on the new approach to attack detection should continue. In this case, development means designing and implementing new functionalities that allow better response to threats and attacks in the network. The first stage of future plugin development is adding the possibility of threat detection based on IPv6. Another idea is to implement a *"refresh"* mechanism – the algorithm could collect information on the network traffic and depending on some constraints new addresses would be added to the *.csv file. This extension of the algorithm could be hard the means of implementation, but it would greatly increase the overall network security.

Intrusion detection and prevention is a very important field of research and will become even more important in the future considering how fast technology development is in today's world. This means that there will always be a need to improve existing and invent new solutions to detect the threats and prevent them. Even if the current solutions are sufficient at the moment, the threats will continue to evolve. Often, phenomena or anomalies in the network seem to be normal behaviour at first glance. However, taking a deeper look, finding correlations with known attacks, or finding differences between user activity and a given event, is the task of the heart of a detection algorithm. It seems likely that the heuristic algorithms based on behaviour analysis will be more and more frequently encountered in the detection of network attacks. This is due to the fact that network attacks are increasingly sophisticated and unpredictable.

Constant development and improvement of tools like *Snort* – for example with an algorithm presented in this paper – allows the networks to become more secure and prepared for future attacks.

Acknowledgements

This work was supported by the *ECHO* project which has received funding from the European Union's *Horizon 2020* research and innovation programme under the grant agreement no. 830943.

References

1 ECHO – European network of Cybersecurity centres and competence Hub for innovation and Operations, accessed June 10, 2021, https://www.echonetwork.eu/.

2 Rafath Samrin and D. Vasumathi, "Review on anomaly based network intrusion detection system," *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Mysuru, India*, 15-16 Dec. 2017, https://doi.org/10.1109/ICEECCOT.2017.8284655.

3 S. Sobin Soniya and S. Maria Celestin Vigila, "Intrusion detection system: Classification and techniques," *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT), Nagercoil, India,* 8-19 March 2016, https://doi.org/10.11 09/ICCPCT.2016.7530231.

4 Karen Scarfone and Peter Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)," Tech. rep. SP 800-94, National Institute of Standards and Technology, 2012.

5 A A Aryachandra, Y. Fazmah Arif, and S. Novian Anggis "Intrusion Detection System (IDS) server placement analysis in cloud computing," *2016 4th International Conference on Information and Communication Technology (ICoICT), Bandung, Indonesia,* 25-27 May 2016, https://doi.org/10.1109/ICoICT.2016.7571954.

6 Snort – Network Intrusion Detection & Prevention System, accessed June 10, 2021, www.snort.org/.

7 NSS Labs. Software tested by NSS Labs (source captured by The Wayback Machine), accessed June 10, 2021, http://nsslabs.com/content/view/15/43/.

8 Doug Dineley, "The greatest open source software of all time," *InfoWorld*, August 17, 2009, www.infoworld.com/article/2631146/the-greatest-open-source-software-of-all-time.html.

9 "SNORT Users Manual 2.9.16. The Snort Project," 2021, http://manual-snort-org.s3-website-us-east-1.amazonaws.com/.

10 Alexander Tzokev, Antonis Voulgaridis, Bartłomiej Gdowski, et al., "Inter-sector prototypes – High-level design," Tech. rep. Deliverable D4.4, ECHO consortium, 2020.

11 Brian Caswell, Jay Beale and Andrew Baker, *Snort Intrusion Detection and Prevention Toolkit* (Syngress, 2007).

[12] Adeeb Alhomouda, Rashid MuniraJules, Pagna Dissoa, Irfan Awanab A.Al-Dhelaan, "Performance Evaluation Study of Intrusion Detection Systems," *Procedia Computer Science* 5 (2011): 173-180.

[13] Martin Roesch, "README.PLUGINS," *Snort FAQ,* accessed June 10, 2021, https://www.snort.org/faq/readme-plugins.

[14] Marcin Niemiec, Rafał Kościej, and Bartłomiej Gdowski, "Multivariable Heuristic Approach to Intrusion Detection in Network Environments," *Entropy* 23, no. 6 (2021), 776, https://doi.org/10.3390/e23060776.

About the Authors

Bartłomiej **Gdowski** is an ICT student at the AGH University of Science and Technology, Poland. Currently, he works on his Master's Degree. His main fields of research include network management and cybersecurity. He participated in the ECHO project, which resulted in several reports and a journal publication.

Rafał **Kościej** is studying Electronics and Telecommunications at the AGH University of Science and Technology, currently working towards his Master's Degree. His main field of research is network security. He has research experience in the ECHO project.

Marcin **Niemiec** was awarded his PhD and PhD Hab. in telecommunications in 2011 and 2019, respectively. Currently, he works as an assistant professor at the AGH University of Science and Technology, Poland. His research interests focus on cybersecurity. He has been involved in numerous European projects (FP6, FP7, and H2020). He is the co-author of over 90 publications.